04 | 问题设计（下）：五个技巧助攻技术问题设计

2021-04-13 四火

技术面试官识人手册 进入课程 >

讲述：四火
时长 18:35 大小 17.02M

你好，我是四火。

在上一讲我们学习了技术问题设计的原则之后，今天我们将继续结合实例，围绕原则来进一步分享一些技术问题的设计技巧，以及一些实践中的注意点。希望在这一讲之后，你可以真正掌握技术问题设计的技巧，真正做到“手边有粮，心中不慌”。

技术问题的设计技巧

下面我将逐条讲解技术问题的设计技巧，这些技巧看似都有各自的独立性，但是其实它是围绕着领域、深度和广度这三个方面展开的，我将给出尽可能具体和完整的例子，来帮助你理解。

从熟悉的领域设计问题

在具体实践方面，这是第一条，也是最重要的一条技巧。**什么样的问题，哪怕在发散性探讨时，你依然觉得尽在掌握？那一定是自己熟悉的。**相信每一个人都有特别熟悉的领域，举个例子来说，我有一位朋友，也是一位面试官，他在亚马逊工作了 9 年，他喜欢和候选人探讨的一个问题是：

> “如果你从头设计亚马逊（或是其它大型电商）的零售业务，你打算怎么做？”

其实这就是一个挺好的问题，话题描述很容易理解，但又足够模糊、足够宽泛，而且可以有很多不同的角度来切入并细化。最重要的是，对他来说，这个领域太熟悉了，基本上不存在明显的思路盲区。

当然，这个问题的领域也必须是“大众”所熟知的，你要是问一个冷门话题，就有可能偏离了考察方向（具体请参考第 3 讲，我在其中做了详细说明）。如果你的工作领域并不为大家所熟知，那么从生活中找问题，或者是从自己的知识领域找问题，也是一个很好的办法，当然，这往往需要多做一点功课。

由浅入深，分层展开

这一条技巧，讲的是“深度”方面，也就是上一讲我们说的“从模糊到清晰，从实际到抽象”原则。还记得我拿来举例的那个问题吗？

> “请你设计一个网约车系统。”

下面我们继续沿着系统设计的路径，分析这个问题深入的过程。

在问题抛出以后，有的候选人能够淡定地做进一步沟通，明确问题，而有的候选人则需要面试官逐步引导——“这个问题具体包括什么？”没错，这正是一个分析和细化需求的过程，需要候选人和面试官一起讨论。

显然，我们不可能在面试中完整实现一个网约车系统，但我们可以专注于某几项核心功能。**我们可以从核心用例的角度来思考，想想都有什么角色，可以通过该系统完成什么功能，比方说**：

1. 乘客可以随时随地叫车；
2. 系统寻找邻近的司机并转发请求，如果无法接单，继续扩大寻找范围；
3. 司机可以抢单，即接受或者拒绝叫车请求；
4. 双方可以刷新自己在地图上的位置，也可以查看彼此当前的位置；
5. 等待期间双方可以查看对方信息；
6. 双方可以查看对方评分，也可以在约车结束后给对方打分；
7. ……

这样梳理后，虽然清楚一点了，但是我们发现其中的内容很多，于是我们可以选择其中最必不可少的一项或几项功能，来跟候选人继续进行探讨，毕竟面试时间有限，贪多嚼不烂。例如，我们就选其中的几项来讨论，其它一概忽略，这就把这个已经简化了的问题进一步简化，于是，问题的范围变得更小了。

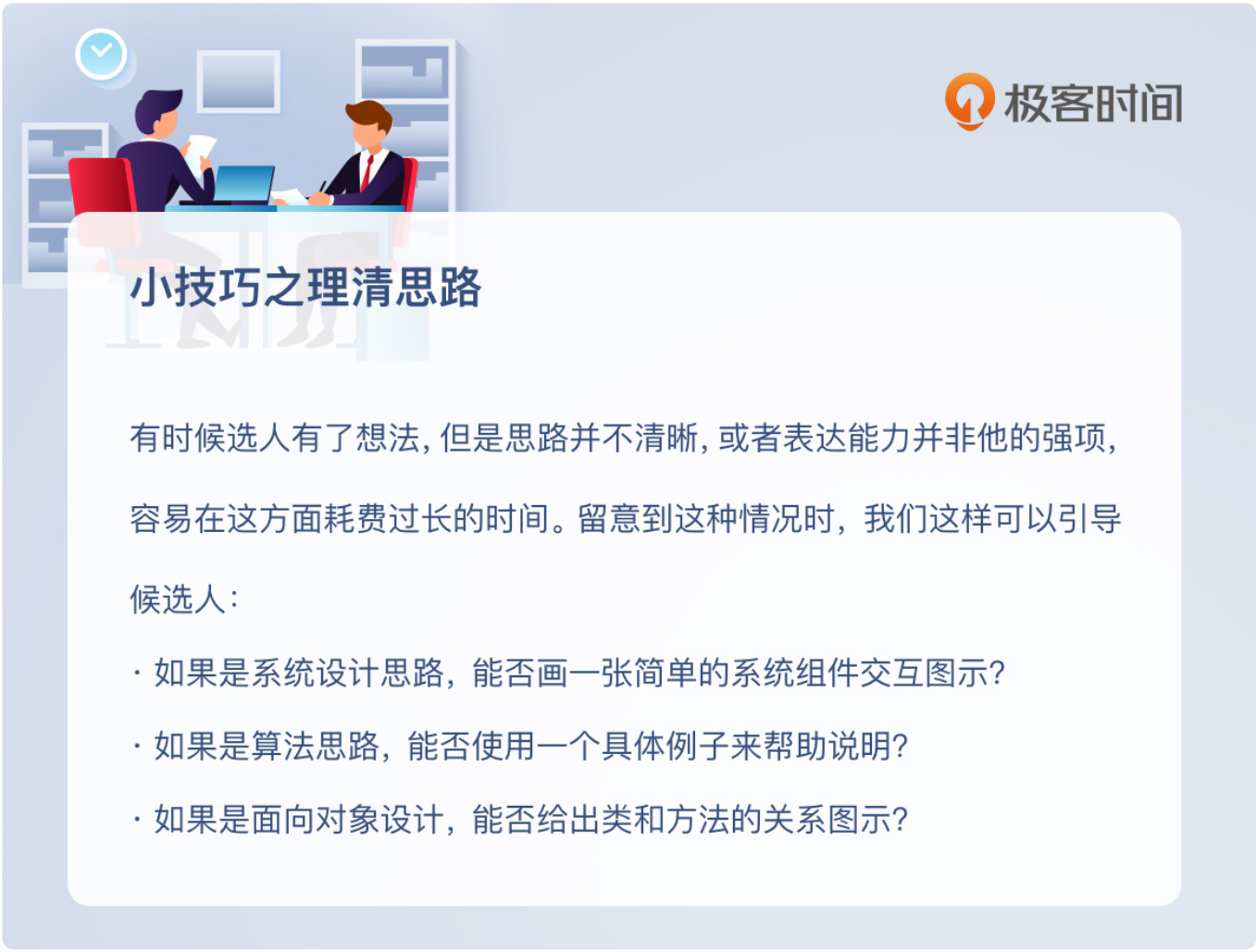
在功能需求明确以后，进入设计步骤前，我们还需要讨论确定几项重要的非功能需求。比如用户量有多少、每日的请求数有多少等等。这些数值不需要多么精准，但是数值的数量级会影响到系统的设计。

在上面的过程中，候选人和面试官一起对问题层层梳理，把一个模糊问题转化为一个简化了的业务问题，只剩下有限的几个需求点要实现。

但是麻雀虽小五脏俱全，功能需求和非功能需求都包括了。下一步，才是根据前面选定的需求点，从技术角度去考虑怎么实现；而技术实现，依然要秉承一样的原则，从模糊到清晰，逐步细化：

1. 整个系统上看，有哪些组件，或者说有哪些 service？（组件的调用和依赖关系）
2. 把系统组件和用户角色放到一起，乘客和司机怎样和系统交互？（交互的时序关系）
3. 每一个 service 都有什么职责，系统的数据存储选用什么技术？（存储层设计）

你看，从上面这几个具体问题也可以看出，这正是在花主要的时间，考察候选人对于系统的理解。



在有了一个大体可行的系统设计后，从中选出一层、一个组件，或是一个机制，进一步细化展开。

比如说，对于存储方面，如果候选人提出使用关系型数据来存储乘客和司机的位置信息，那么我们可以通过这样的思路来展开：

1. 你打算设计怎样的表结构来存储这样的信息？
2. 这样的方案是否存在隐患，在产品上线后可能发生怎样的问题？
3. 你打算怎样解决这样的问题？

设定“踮踮脚能够到”的最高难度

下面来谈谈问题的难度。

很显然，不同的候选人，能力是不同的，对于同一个问题能够完成的进展也是不同的。我前面已经说了，问题的深入要有层次，逐步细化问题、抽丝剥茧，最后落实到一个短时间

内就能完成的局部实现上。

如果留意到问题对于候选人较为困难，面试官需要动态调控，目的是降低难度，方法可以是给出问题提示、主动减小问题范围、弱化问题要求，甚至直接推进问题分析。而如果问题对于候选人来说过于简单，那就要压缩流程中时间的分配，更多地把时间放到问题解决之后，对于进一步改进等等的“follow-up”的讨论上面。

举个例子，如果一个算法问题是，使用非递归算法实现二叉树的前序和后序遍历，而候选人看起来缺乏头绪，那么我们可以：

给出问题提示：“二叉树是怎样的一种数据结构？前序和后序遍历的定义是什么？”；

减小问题范围：“我们先来只实现前序遍历吧！”；

弱化问题要求：“如果我们可以使用递归来实现呢？”；

推进问题分析：直接介绍前序或者后序遍历，使用非递归来实现的思路。

一句话，通过暗示也好，明推也好，不断深入并击破难度递增的每一层问题挑战，并且**最理想的效果，是最终这个迷你项目的最高难度，恰恰是候选人“踮踮脚能够到”的难度，既不会过度简单，又不会难不可达。**

这就有点像打电脑游戏，太简单了没意思，候选人也会觉得这家公司和团队不怎么样；太难了则又可能对候选人打击太大，或者导致他无法完成整个问题的解决流程。我们可以通过上面的四种办法，引导候选人理清思路，直到可以开始动手编码的程度。

这个动态调整的过程是需要通过经验的积累，才能驾驭自如。但是我们从一开始，就可以树立这样的目标，我们希望不同程度的候选人，都可以“来时开开心心，走时不住回味”。

面试的过程，如我们之前所说，考察是双向的，我们既要考察候选人的情况，也要给候选人留一个好印象。

持续收集数据，调整你的问题

对于前面提到的“问题的解决流程要完整”，和“抵达‘踮踮脚能够到’的难度”，不知你是否会觉得听起来很难做到，这需要比较高的经验和技巧。

没有错，这也是为什么我们需要“持续收集数据，调整你的问题”。就好像要把任何一件复杂的事情做好，都要经过千锤百炼一样，一个问题设计好了以后，我们要反复地在实践中使用它。

一方面，可以**不断地获取数据，这样的数据可以帮助你建立起属于自己的“坐标参考系”**，在评估一个新的候选人的时候，我们知道他大概处于一个什么位置；另一方面，又可以根据结果，反向地调整、改进，甚至替换问题，包括可以根据情况适当地增大或者减小问题的范围等等。

举个例子，我认识一位面试官，他在考察算法的面试中，使用一道有关无向图求路径的问题，如果候选人对于无向图的基本知识缺乏了解的话，它就是一道挺难的题目了，毕竟，图的表示和路径求解，并不包含在我们最常见的数据结构和算法里面。这个题目他用了好几次，但是发现候选人普遍“没有思路，答不上来”，于是就不再使用这个题目了。

我的一个经验是，在一个问题（包含确定的某一考察角度）被用了 5 次以上之后，基本上这个问题本身就成熟得多了。

使用一个你自己设计的问题，还会给候选人带来一定的“新鲜感”，做这个迷你项目，解决一个新颖、有趣的问题，而不是互联网上一搜一大把那些。感觉这是一个信手拈来的事情，可他所不知道的是，这个问题是面试官精心准备的，也是在实践中锤炼过的。

拓宽广度，给问题设置多角度延伸

这一条技巧讲的是“广度”，其实就是我们上一讲的“不止一个考察角度，不止一个解”原则。

这次拿我自己来举例，我拿这样一个问题，问了好几十人：

> “如果你所在的团队拥有一个重要的 service，它的平均请求响应时间为 1 秒，有一天你部署了软件新版本后，留意到它的平均请求响应时间异常地增大到了 10 秒，你该怎么做？”

这显然是一个实际的问题，而且可以是一大串不同问题的引子，根据考察的需要，我们可以把它延伸出去，进行各种追问。

延伸一——**考察对用户体验、对工程的理解**：第一时间我们该怎样做，才能把对用户的影响降到最低？我们可能会谈到回滚，那么我们就可以讨论 Ops。怎样避免未来再出现这样的问题？这就可以讨论软件工程的流程，讨论质量保证，讨论 CI/CD，讨论灰度发布等等。这个延伸可以考察软件工程流程的方面，考察 Engineering Excellence 的方面。

延伸二——**考察监控和告警系统设计**：我们该怎样争取在第一时间知道我们的系统出了问题？这就可以谈论软件监控和告警，我们需要对怎样的数据进行监控和告警？怎样采集数据，怎样发送数据，即怎样设计告警系统？这个延伸可以继续细化到一个监控系统设计的问题，或是做代码层面的考察。

延伸三——**考察问题排查思路、系统的理解**：我们该怎样去定位问题，问题可能出现在哪一个层面？

比如，可能 service 根本没事，只不过客户端出了问题，也可能网络有了问题，还可能就是 service 出的问题。而 service 出的问题，又可能有哪些原因造成，可以采用怎样的策略，去一层一层从高到低去定位问题所在？这一个延伸可以继续考察候选人对于系统的理解，尤其是通过一个请求响应的过程，考察他对于系统全栈的理解。

延伸四——**聚焦系统某一层**：通过细化到某一层来进一步展开，这部分的思路就很多了，根据需要我们可以选择几个常用的切入点。

比方说，如果问题是关系数据库的语句过慢造成的，这里我们就可以进一步聚焦，讨论一些常见的关系数据库性能问题出现的原因，以及相应的解决思路。如果是应用层面的问题，比如应用的内存不足，那么可以从这个角度进一步展开。

延伸五——**聚焦系统限流问题**：如果问题是恰巧由于该时间点某一客户端在短时间内发送过量的请求，系统无法及时处理而导致的拖慢，那么我们就又可以从流量控制的角度展开。流量控制的问题也是属于经典的系统问题之一，既可以从系统设计的角度考察，也可以从代码设计的角度考察。

你看，一个模糊的实际问题，可以拿过来做多种方向的延伸，而这些延伸之间又具备密切的联系。我们可以根据需要，在面试中选取其中一个延伸的角度来进行考察。

平衡考察的深度和广度

如果我们总是能把技术考察，从一个简略、模糊的实际问题开始，逐步深入，到最后以一个完整的问题解决过程收尾，那么几乎就可以说，我们就做到了深度和广度的兼顾。

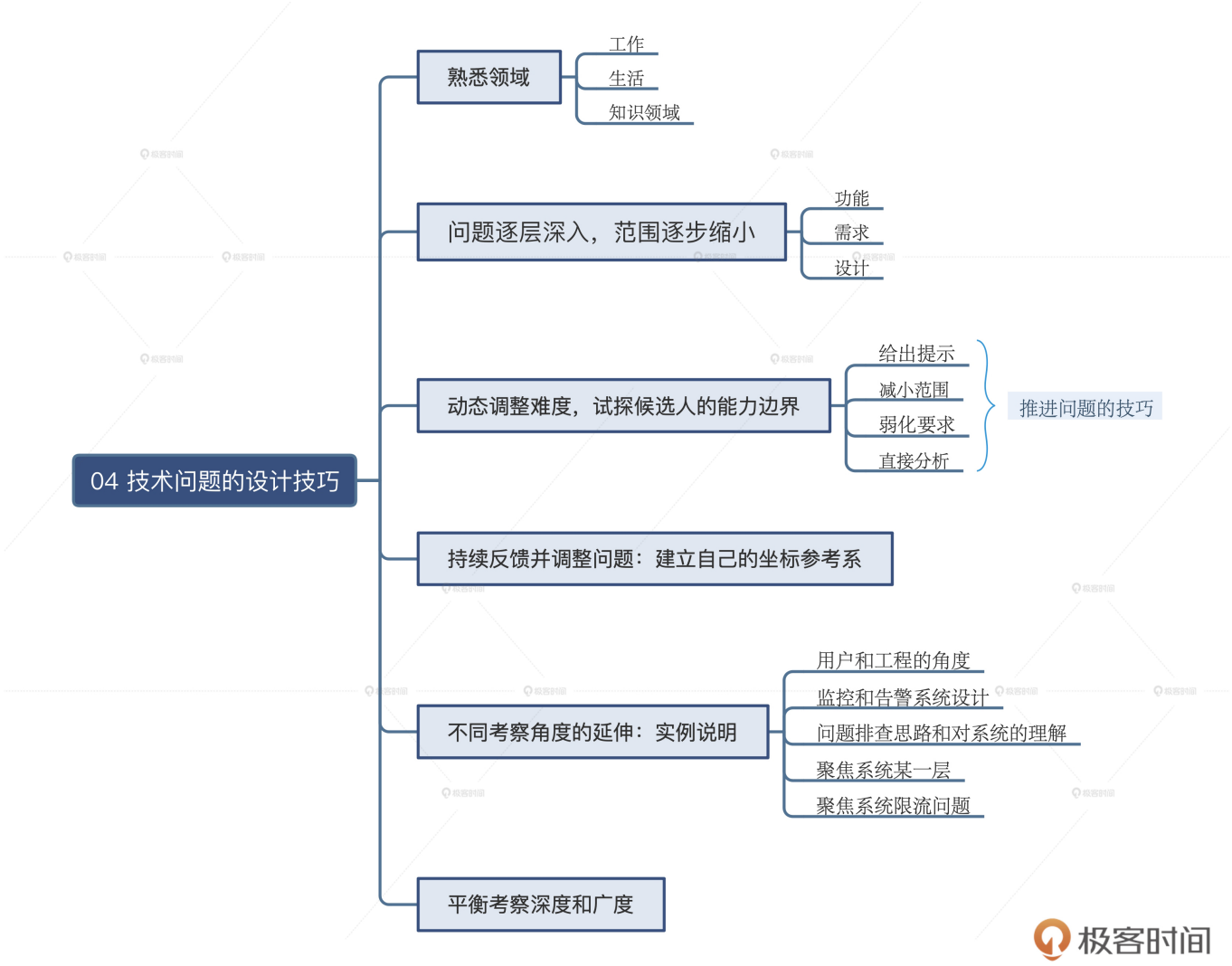
我曾经两次提到过，为什么我不推荐面试中使用简单粗暴的纯算法题？说到底就是一个考察面的问题，单纯地考察算法题，就是一个可以具备足够的考察深度，但是很容易丢失考察广度的例子。

反过来也一样，我下面来说说覆盖广度而丢失深度的情况。有一种面试风格，是让候选人谈论一个他自己最熟悉，或是最自豪的一个项目，然后从中抓住某些点不断深挖。这本身听起来似乎也是一个挺不错的考察思路，和我们前面谈到的面试官去准备问题的思路，恰好相反——由候选人来主导这个过程。

但是现实中，对于经验尚浅的面试官，我一般不太推荐上面的方式，主要原因是，在实践中这个方式对于面试官的素质要求非常高：如果候选人谈到的项目或者系统，面试官比较熟悉，那一切都好说，但是如果不熟悉呢，面试官就很难把握住这里的关键，容易被候选人牵着鼻子走，可能这一轮面试候选人一直在泛泛地说，谈到了许多方面，但是面试官也提不出什么一针见血的问题。

总结与思考

在上一讲谈论的技术问题设计原则之后，今天我们结合了几个实际例子，讨论了几个技术问题的设计技巧，相信你通过今天的内容，在之前大致思路的基础上，收获到了一些可以用于实践的小窍门。



在文章的最后，我想提一个问题，你能否分享一下，你觉得行之有效的，在面试中考察候选人的小窍门呢？

好，我是四火，我们下一讲见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | 问题设计（上）：三大原则理清面试考察方向

下一篇

精选留言 (2)

写留言

lihp
2021-04-17

从候选人熟知的技术问题开始，向下深入，探索底层的理解程度；向上拓展，从具体的问题到实际工作中的解决经验和方案构建思路。
其中一个问题是让候选人写一段简单的功能代码（20行左右），向下讨论代码的执行过程，函数调用过程，内存模型等；向上代码本身的美观、模块化，接口设计的原则，不同模块接口之间调用组合，接口后期扩展的处理方案，模块内部异常的排查思路或手段，...
展开

作者回复: 嗯，这个思路我觉得是很不错的。
当然，和上面那个留言的回答一样，面试官也要清楚，这个思路具备哪些局限性，又覆盖了哪些考察角度。

3

fakefish
2021-04-13

从前端角度有个问题是webpack怎么打包，大部分候选人知道可以分包，但会深入问
1、打包完文件几种hash区别
2、分包之后文件怎么部署
3、问题2引出浏览器缓存
3、问题2引出浏览器并发下载限制...
展开

作者回复: 思路不错，但是我想说明两点：
1. 这个问题有一定局限性，（1）只能是针对前端工程师；（2）这是一个很好的技术问题，考察了知识面、对于背后机制的理解，但较少对于problem solving的覆盖

所以使用这样问题的时候，需要清楚地理解它的局限性和考察角度。

3